垃圾回收概述

一、常见术语

1.1 并行

两个或者多个事件在同一时刻发生，在现代计算机中通常指多个处理器上同时处理多个任务。

1.2 并发

两个或者多个事件在同一时间间隔发生，在现代计算机中一台处理器"同时"处理多个任务，这些任务只能交替运行，从处理器的角度看任务是只能串行，但从用户的角度看，这些任务是“并行”执行的，实际上是处理器根据策略不断切换知行合一这些“并行”的任务。

所以，并行和并发是从处理器的角度出发。但是在垃圾回收领域或者jvm领域，并行和并发在jvm中被重新定义了。

1.3 jvm中的并行

多个垃圾回收相关的线程在操作系统之上并发运行，这里的并行强调的是只有垃圾回收线程工作，java应用程序终止(或称为暂停)，因此类似ParNew工作的时候，一定发生了STW。

1.4 jvm中的并发

垃圾回收相关的线程并发运行（如果启动多个线程的话），同时这些线程会和java程序并发运行。

1.6 STW（stop-the-world）

停止一切，jvm中指停止一切java应用的线程。

1.7 安全点

jvm在执行一些操作的时候需要STW，但并不是任何线程在任何地方都能进入stw。
设置安全点的目的，当线程进入到安全点时，线程会主动停止。

1.8 Mutator

通常指我们的java应用线程。含义是可变的。这里的含义是因为线程运行，导致了内存变化。GC中通常需STW才能使Mutator暂停。

1.9 记忆集

Remember Set.简称Rset。记录不同代对象之间的引用关系。

1.10 Refine

在G1中主要指处理RSet的线程

1.11 Evacuation

转移、撤退、回收。
简称Evac。在G1中指的是发现活跃对象，并将对象复制到新地址的过程。

1.12 回收(Reclaim)

分区对象已经死亡或者已经完成Evac，分区可以被jvm再次使用

1.13 Closure 闭包

jvm中的一种辅助类，类似于iterator。通常提供了对内存的访问。_

1.14 GC ROOT

垃圾回收的根，jvm垃圾回收的过程中，需要从gcroot出发，标记活跃对象，确保正在使用的对象在垃圾回收之后都是存活的。

1.15 根集合（Root Set）

jvm的垃圾回收过程中，需要从不同的gc root出发，这些gc root有线程栈、monitor列表、jni对象等等。这些根就构成了根集合。

1.16 Full Gc

简称FGC,整个堆的垃圾回收动作。通常FGC是串行的，G1的FULL GC不仅有串行实现，在JDK10中还有并行实现。

1.17 再标记

Remark.指的是并发标记算法中，处理完并发标记之后，需要更新并发标记中Mutator变更的引用，这一步一般是需要STW的，不然标记过程是无法终止的。

二、回收算法概述

垃圾回收(Garbage Collection ,GC) 指的是程序不用关心对象在内存中的生存周期，创建之后只需要使用，不用关心何时释放以及如何释放，由jvm自动管理并释放。

垃圾回收主要是针对堆空间回收，主要分为两大类：

- 引用计数法
  堆内存中分配对象时，会给对象分配一个额外的空间，维护一个计数器，新增对象了加之。如果一个引用关系失败则减少，当为0的时候，说明已经废弃，处于不活跃状态，可以被回收。它需要解决循环依赖的问题，python就是这个。
- 可达性分析法
  将根集合作为七点，从这些节点往下搜索，搜索走过的路径称为引用链，当一个对象没有任何引用链访问的时候，此对象是不活跃的，可以被回收。

jvm采用了可达性分析法。

垃圾回收算法也在不断演进，主要有以下分类：

- 它主要分为赋值、标记清除、标记压缩
- 在回收方法上又可以分为串行、并行、并发
- 内存管理上又分为代管理、非代管理

2.1 分代收集算法

分代管理就是把内存区域划分成不同的区域管理，思想来源就是：有些活的时间段，有些活的时间长，把时间短的放在一个区域，时间长的放在一个区域。

可以针对不同区域选择不同算法，加快收集效率。
假定分为新生代、老生代。易死亡的放在新生代，通常采用复制算法回收；预期存活时间长的放在老生代，采用标记清除算法。

2.2 复制算法

具体实现有很多种，可以使用两个区，也可以使用多个区。两个分区的时候内存利用率只有50%，多个分区的时候可以提高内存利用率。

假定把空间分为1个新生代（分为三个区，一个Eden、一个Survivor0、一个Survivor1）一个老生代的过程。

普通对象创建的时候再Eden区，S0和S1分别是两个存活区。第一次垃圾回收之前，S0和S1都是空的，在垃圾收集之后，Eden和S0里面的活跃对象都放入S1。

回收之后，会继续运行并产生垃圾，在第二次运行前Eden和S1都有活跃对象在垃圾回收之后，Eden和S1里面的活跃对象都被放入S0区，一直这样循环收集。

2.3 标记清除

从根集合出发，遍历对象，把活跃对象入栈，并依次处理。此处的遍历，可以使深度优先、也可以广度优先，通常采用深度优先，节约内存。标记出活跃对象之后，就可以把不活跃对象清除。

标记清除算法，需要额外数据结构记录可用空间，在分配的时候从何可以空间链表中寻找能容纳的空间。

缺点
最大缺点就是内存碎片化。

2.4 标记压缩

为了解决标记清除算法使用过程中内存碎片化的问题，除了上述的标记动作之外，还会把活跃对象重新整理从头开始排列，减少内存碎片。

垃圾回收算法比较

算法	优点	缺点
分代	组合算法，分配效率高， 堆的使用效率高	算法复杂
复制	吞吐量大，分配效率高，无碎片	堆的效率低，需要额外的空间，需要移动对象
标记压缩	堆的使用效率高，无碎片	暂停时间长，缓存不友好 (对象移动后顺序关系没有了)
标记清除	不需要移动对象，算法简单	内存碎片化，分配慢（需要找到合适的空间）

三、垃圾回收器

为了达到最大性能，基于分代管理和回收算法，结合回收时机，jvm实现的垃圾回收机器：
串行回收、并行回收、并发标记回收（CMS）、垃圾优先回收。

3.1 串行回收

单线程。在回收的时候Mutator需要STW，新生代通过复制算法，老生代采用标记压缩算法。

3.2 并行回收

多线程，在回收的时候Mutator需要暂停，新生代采用复制算法，老生代采用标记压缩算法。

3.3 并发标记回收

并发标记回收CMS，整个回收期间划分成多个阶段：初始阶段、并发标记、重新标记、并发清除。初始标记、重新标记的阶段需要暂停Mutator，在并发标记和并发清除期间可以和Mutator并发运行。通常用于老生代，新生代采用并行回收。

3.4 垃圾优先回收

垃圾优先回收（G1）致力于多CPU和大内存服务器上对垃圾回收的提供软实时目标和高吞吐量。G1的设计和前面提到的三种不一样，他在并行、 串行、CMS针对堆空间的管理方式上都是连续的。

连续的内存导致垃圾回收时间时收集时间过长，停顿时间不可控。G1把堆拆成一系列的分区，这样在一个时间段内，大部分的垃圾收集操作只站队部分分区，而不是整个堆或者整个代。

G1里，新生代就是一系列的内存分区，因此意味着不用再要求新生代是一个连续的内存块。类似，老生代也是由一系列分区组成，也就不需要在JVM运行时考虑哪些分区是新生代，哪些分区是老生代。

事实上，G1的运行状态是：映射G1分区的虚拟内存随着时间的推移在不同的代之间切换。

G1新生代的收集方式是并行收集，复制算法。与其他的jvm垃圾收集一样，一旦发生一次新生代回收，整个新生代都会被回收，这也就是新生代回收（Young GC）。
G1和其他的垃圾回收期不同的地方在于：

- G1会根据预测时间动态改变新生代大小
- G1老生代的垃圾回收方式和其他的jvm回收老生代处理有着极大的不同。G1老生代的收集，不会为了释放老生代的空间堆整个老生代回收。相反，任意时刻只有一部分老生代分区会被回收，并且这部分老生代分区将在下一次增量回收的时候与所有新生代分区一起被收集。这就是所谓**混合回收（Mixed GC）**。选择老生代分区的时候，优先考虑垃圾多的分区，这也是G1的由来。

大对象
待分配的对象大小超过一定的阈值之后，为了减少这种对象在垃圾回收过程中的复制时间，直接把对象

分配到老生代分区，而不是新生代分区中。

从实现角度，G1算法是复合算法，吸收了以下算法的优势：

- 列车算法，对内存进行分区
- cms，对分区进行并发标记
- 最老优先，最老的数据优先收集